GeomLab – Exploring Computer Science

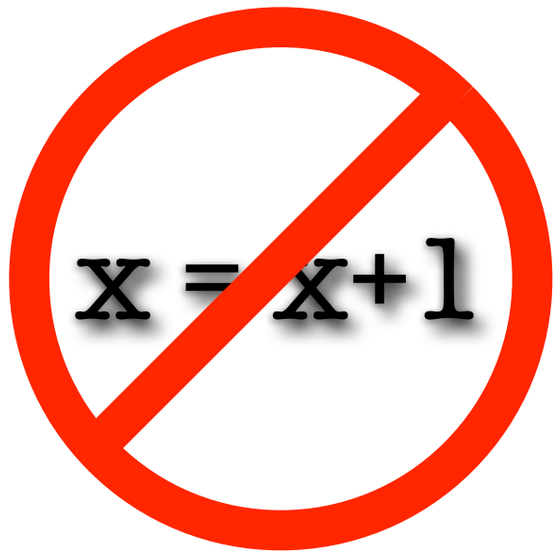
Michael Spivey

Department of COMPUTER SCIENCE
UNIVERSITY OF OXFORD

What's programming like?

The old-fashioned view:

- A program is a list of instructions.
- Algorithms are designed with flowcharts.
- Computer memory is boxes containing numbers.
- It's all ones and zeros!



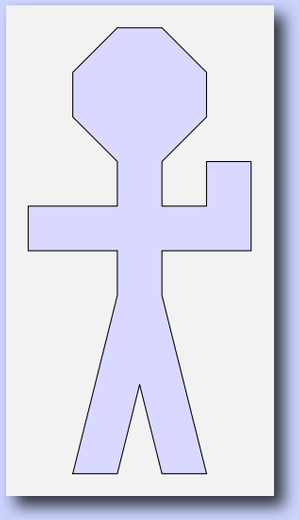
What's it really like?

A modern view:

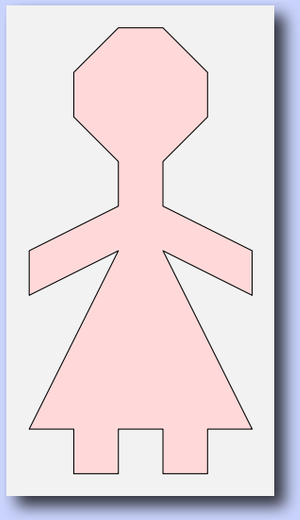
- A program is an artificial world.
- Objects behave according to precise rules.
- Combining simple behaviours can let a complex whole emerge.
- We can use things without knowing how they are made.

A world of pictures

man

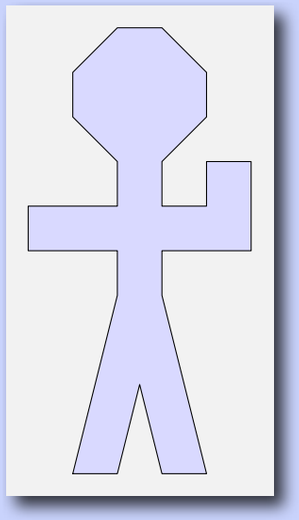


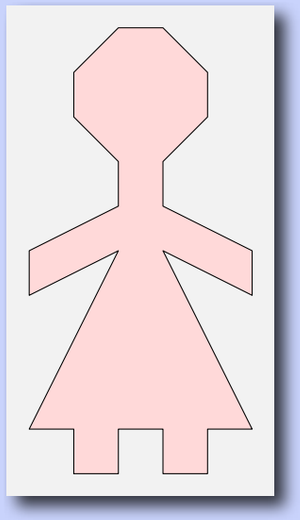
woman



One be$ide another

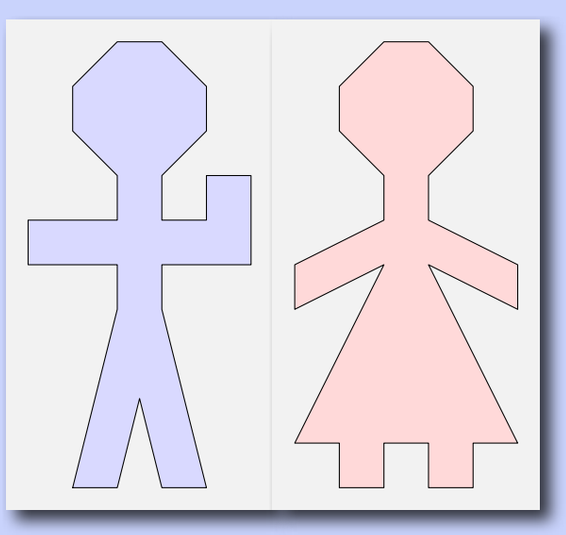
man $ woman





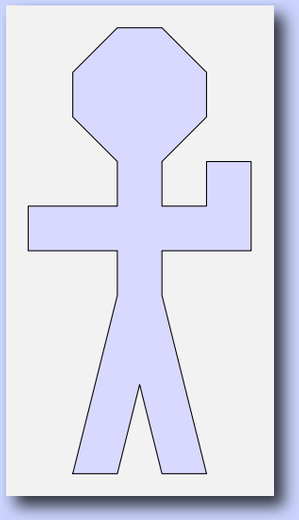
One be$ide another

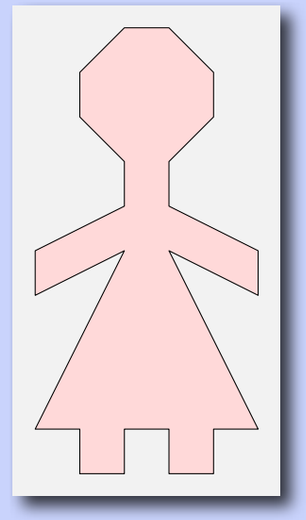
man $ woman



One &bove another

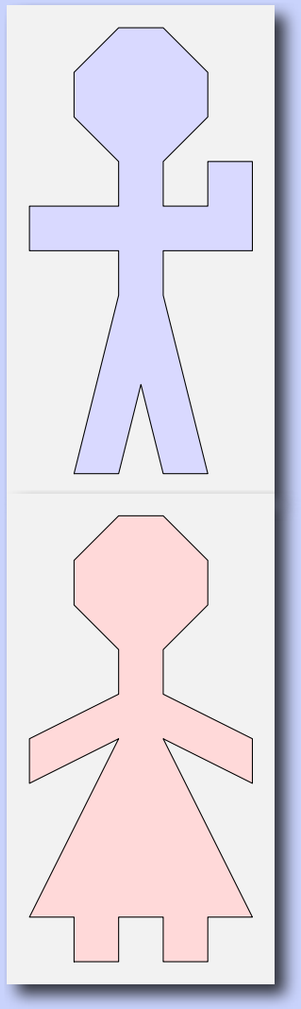
man & woman





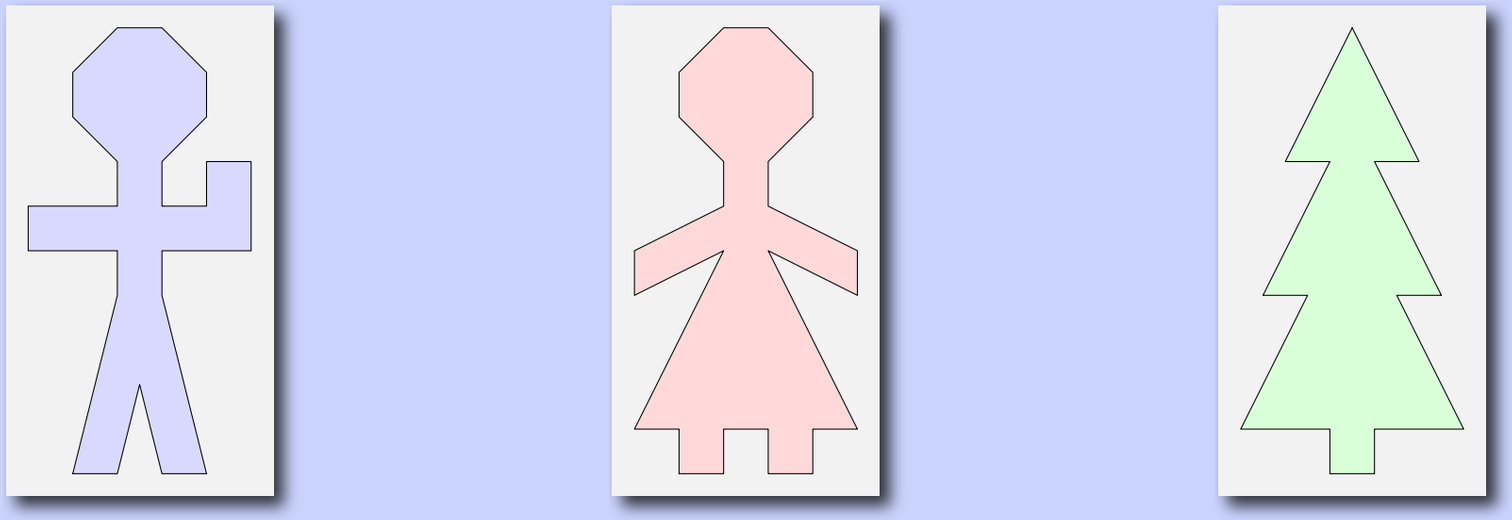
One &bove another

man & woman



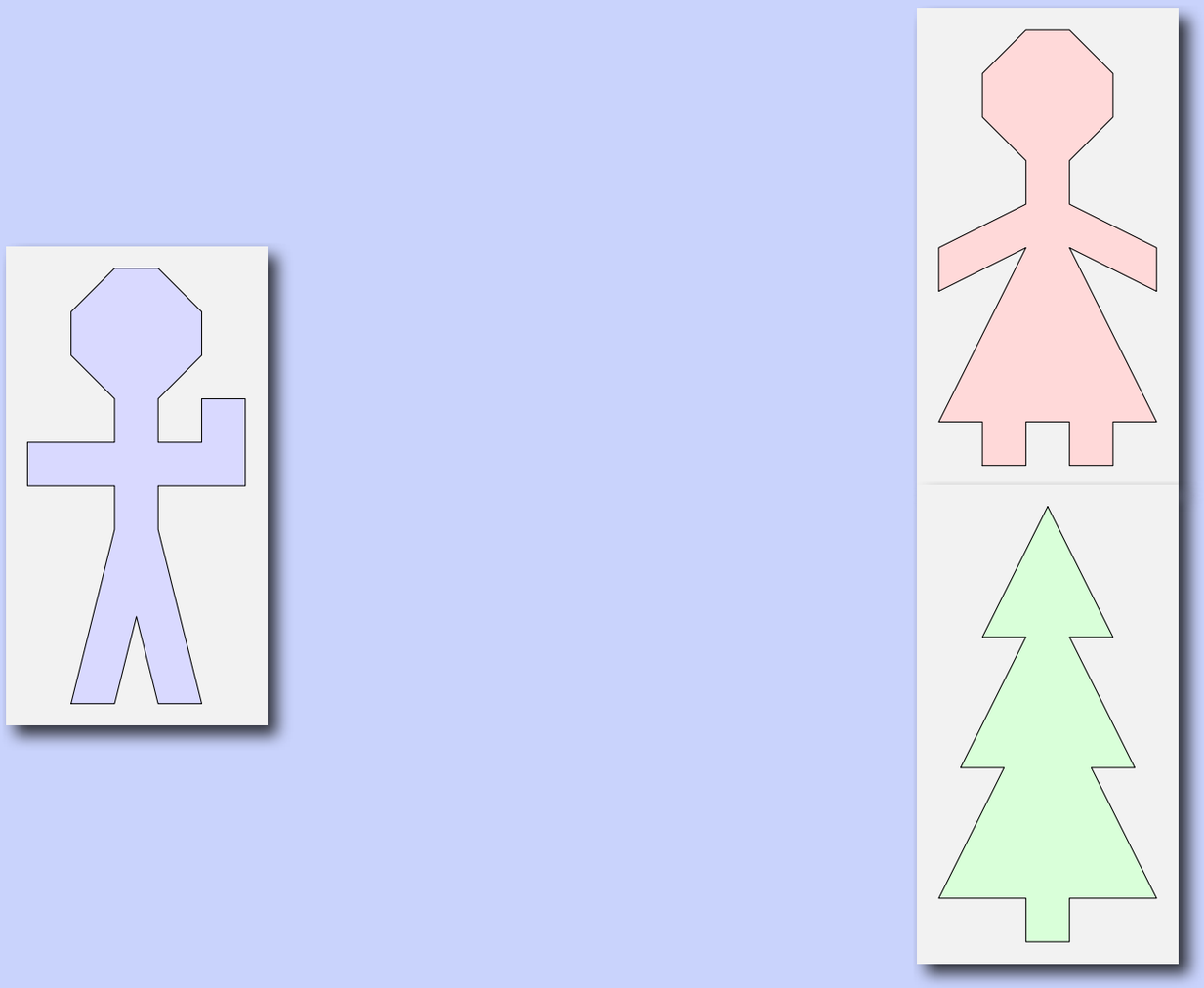
Combining operations

man $ (woman & tree)



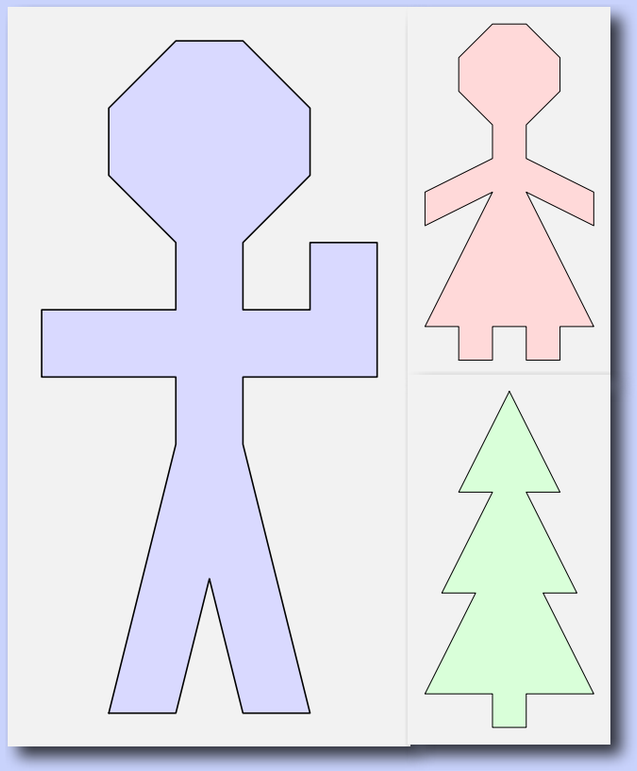
Combining operations

man $ (woman & tree)



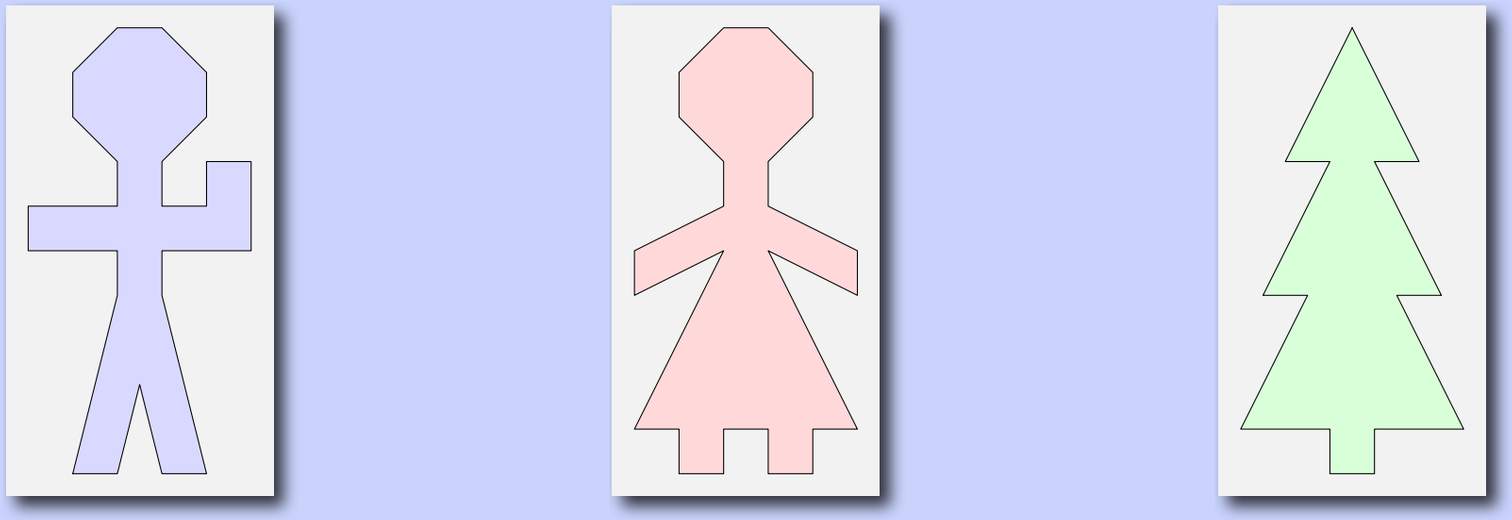
Combining operations

man $ (woman & tree)



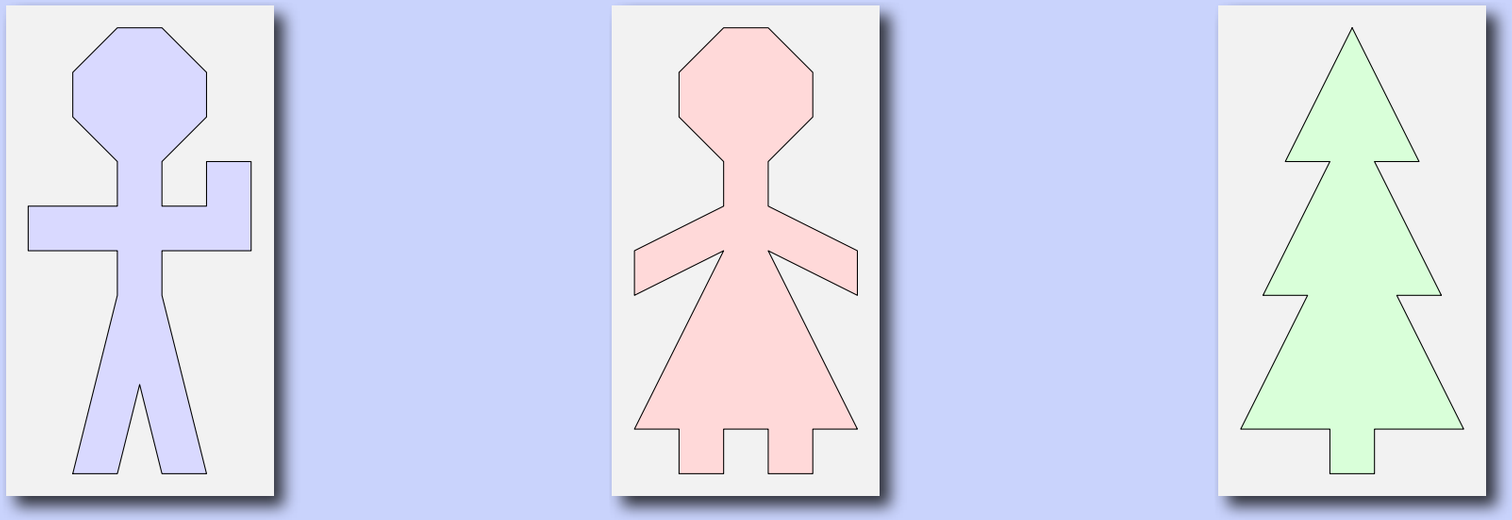
Order matters!

man $ (woman & tree)



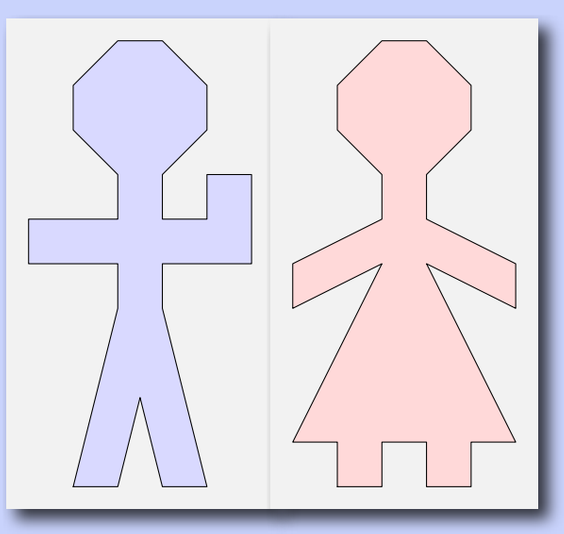
Order matters!

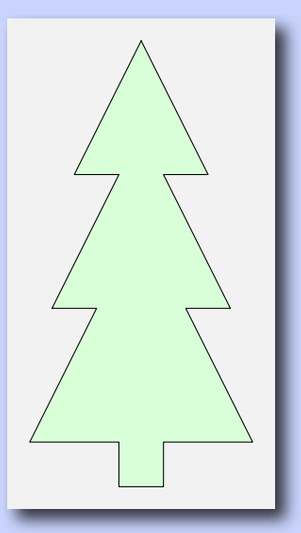
(man $ woman) & tree



Order matters!

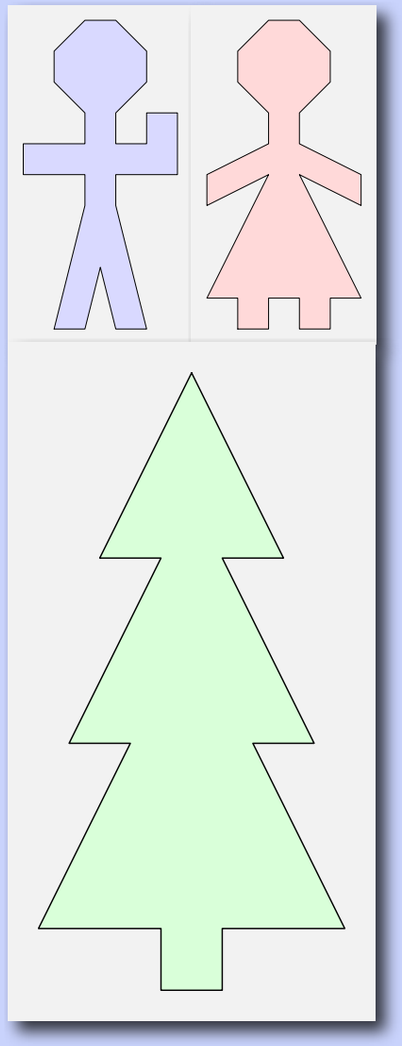
(man $ woman) & tree





Order matters!

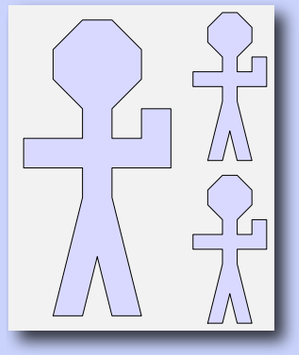
(man $ woman) & tree



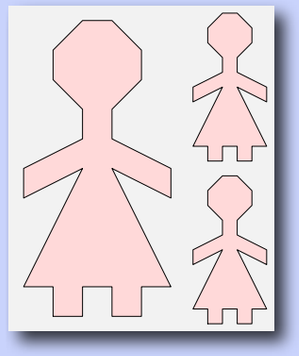
Defining functions

define f(x) = x $ (x & x);

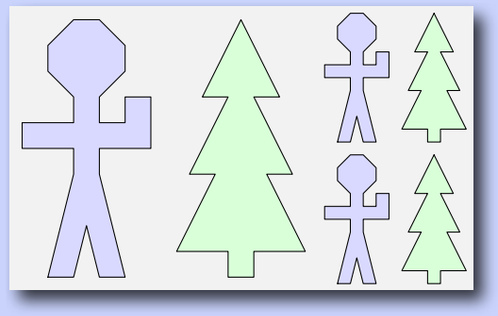
f(man)



f(woman)

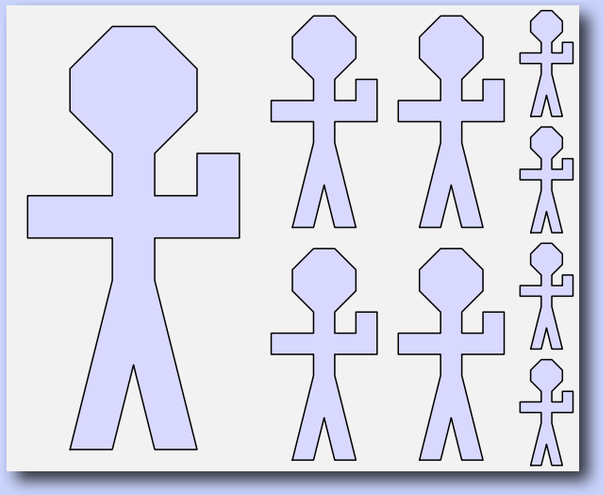


f(man $ tree)

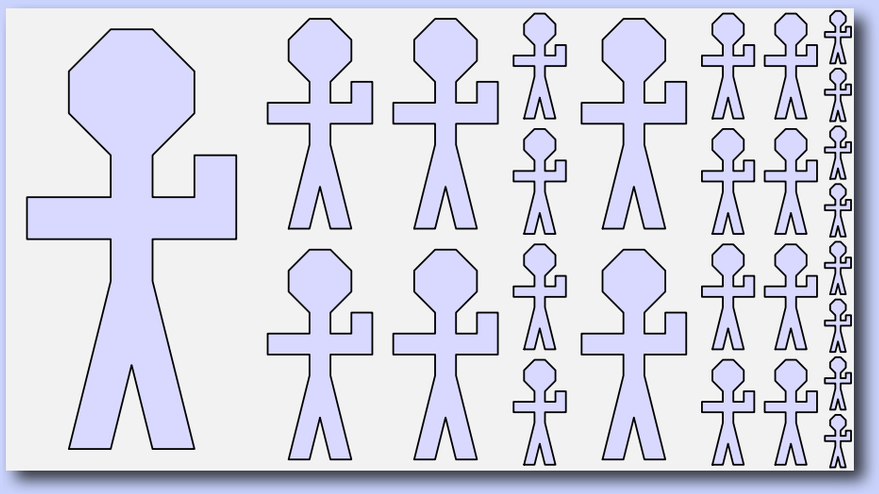


Composing functions

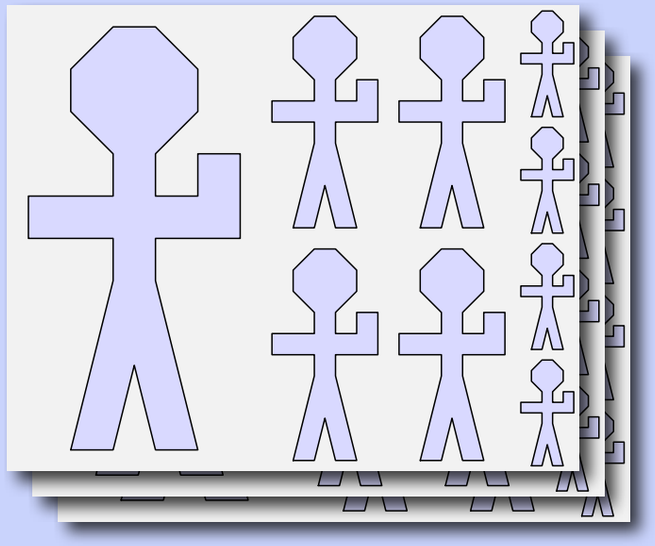
f(f(man))



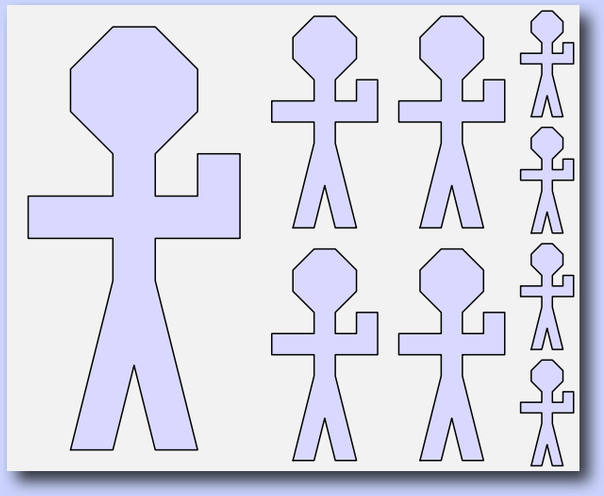
f(f(f(man)))

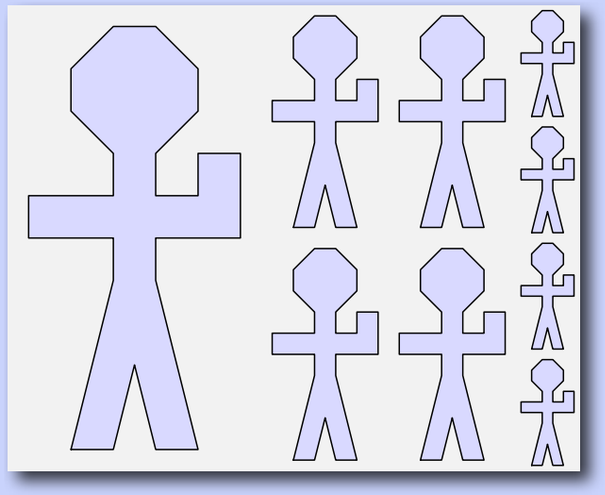


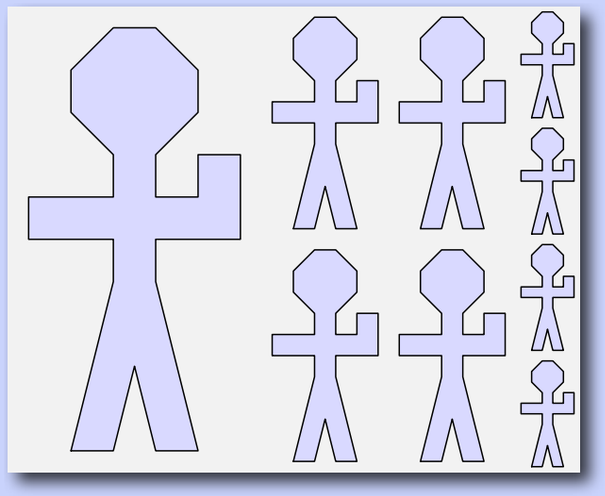
Why so complicated?



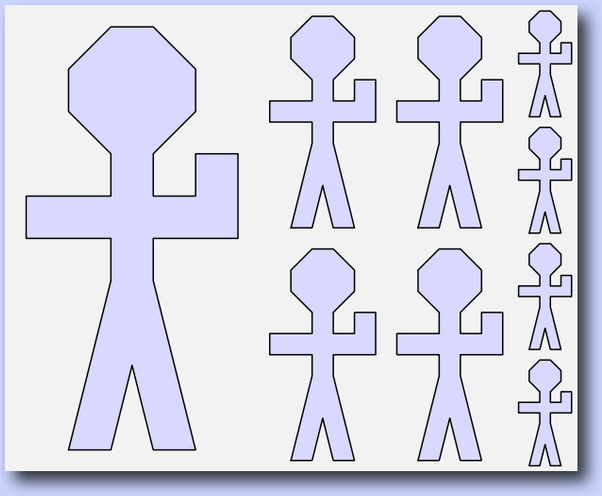
Why so complicated?

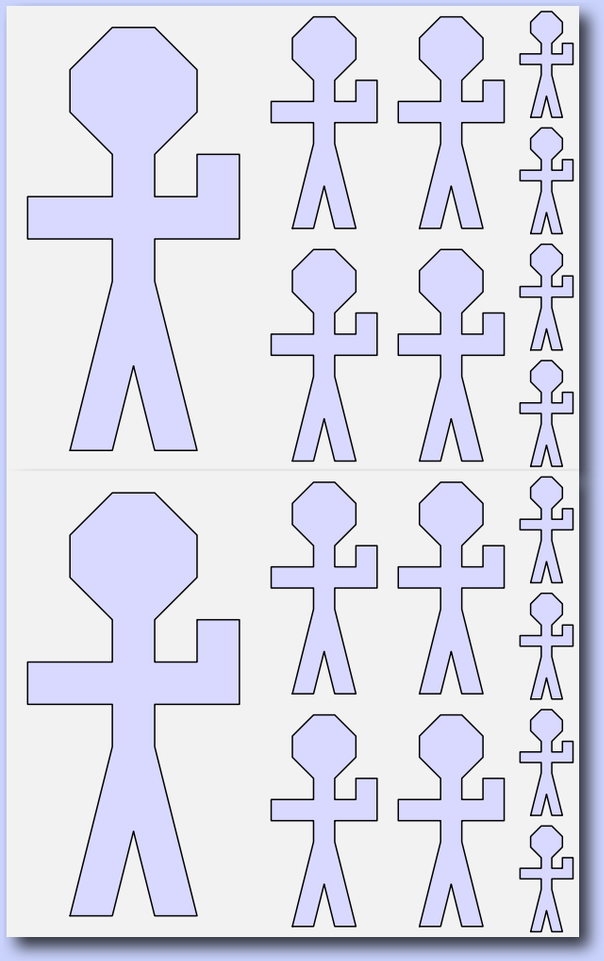




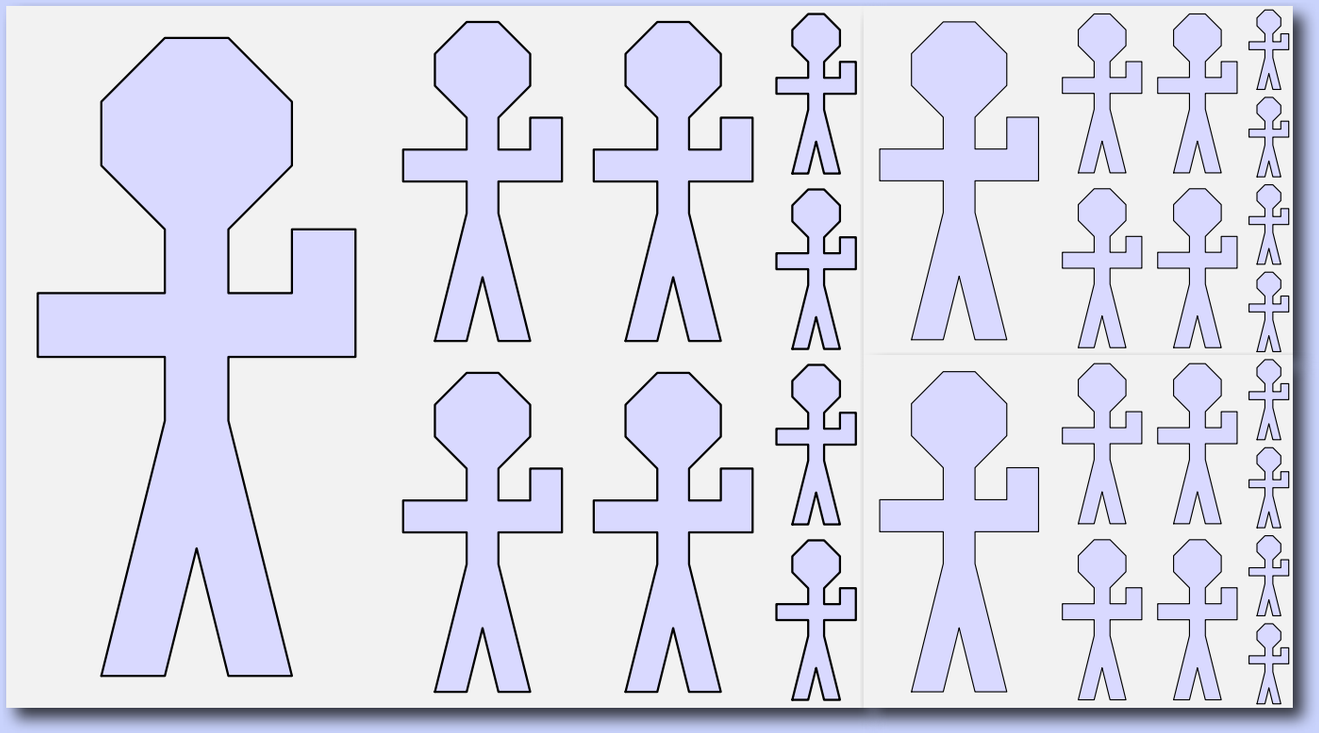


Why so complicated?





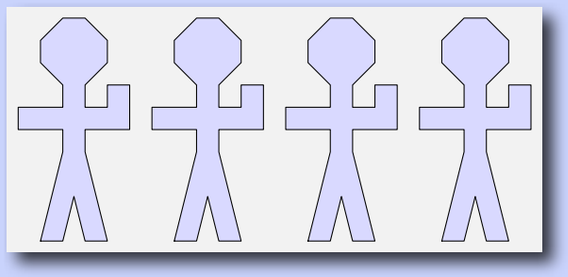
Why so complicated?



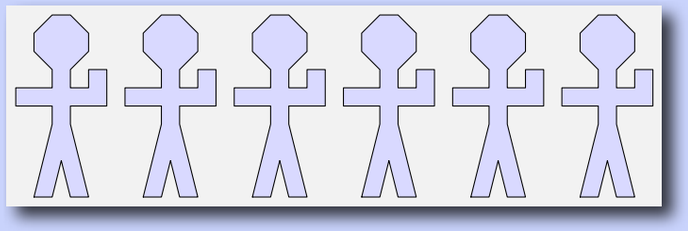
Recurrence and recursion

```
define row(1, p) = p
  | row(n, p) = p $ row(n-1, p) when n > 1;
```

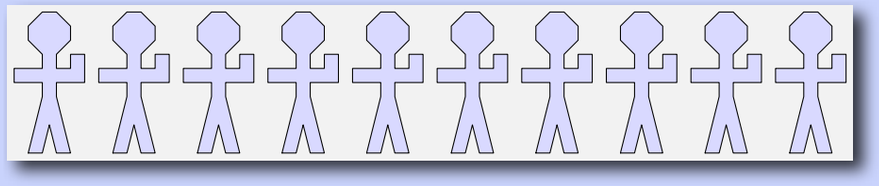
row(4, man)



row(6, man)



row(10, man)



Calculating the result

```
define row(1, p) = p
  | row(n, p) = p $ row(n-1, p) when n > 1;
```

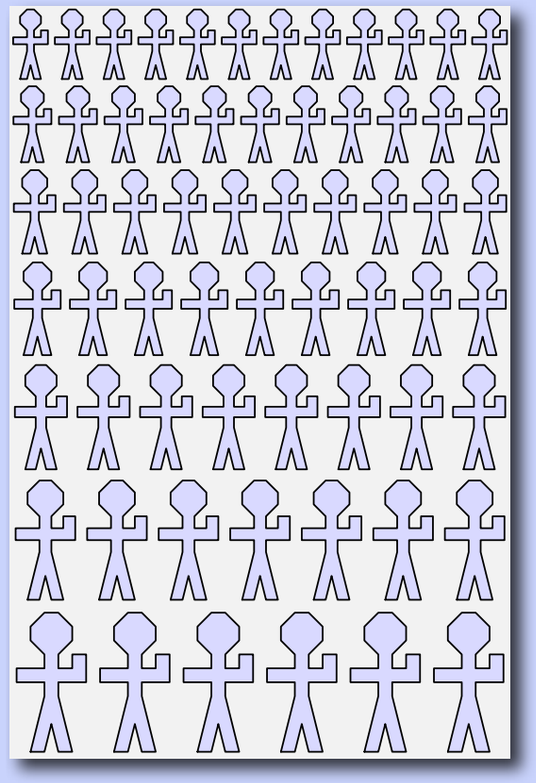
So:

```
    row(4, man)
= man $ row(3, man)
= man $ man $ row(2, man)
= man $ man $ man $ row(1, man)
= man $ man $ man $ man.
```

Drawing a crowd

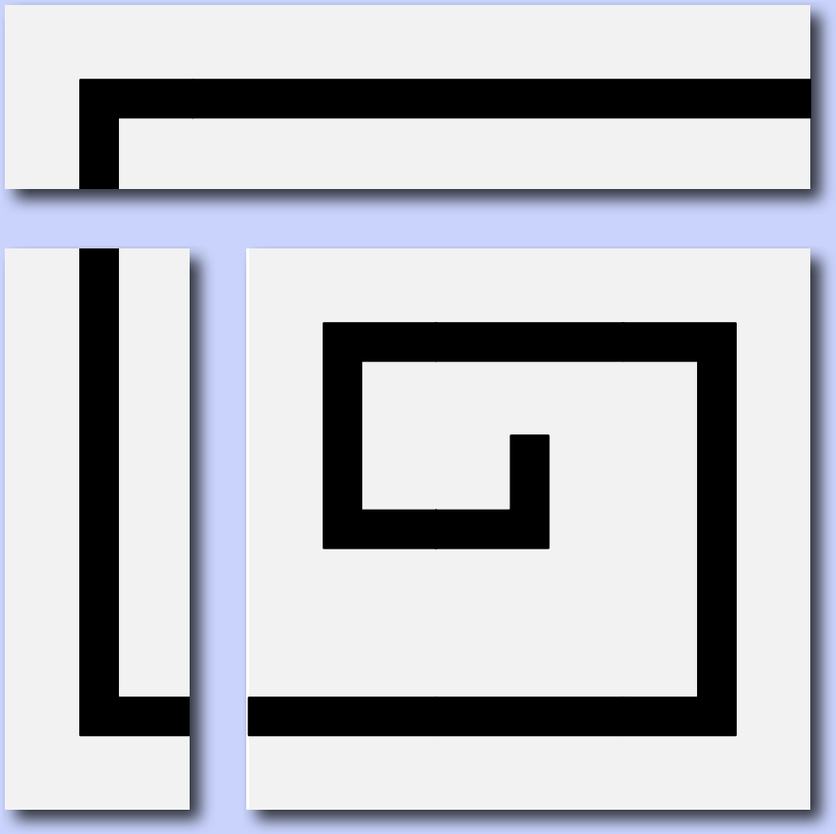
```
define crowd(a, a, p) = row(a, p)
  | crowd(a, b, p) = crowd(a+1, b, p) & row(a, p) when a < b;
```

crowd(5, 10, man)



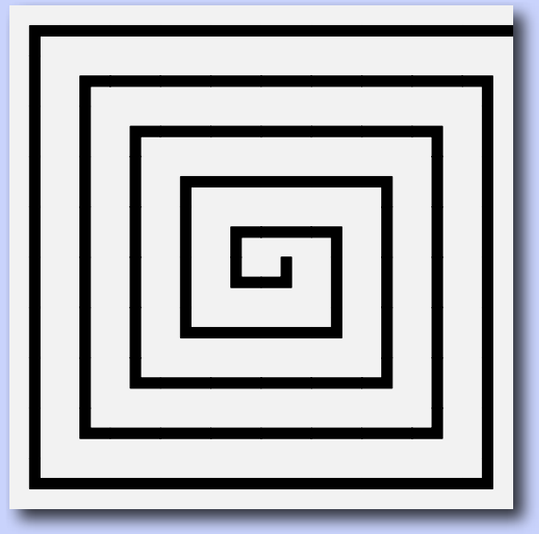
Another recursive pattern

```
define spiral(1) = bend
  | spiral(n) = arm(n) & (rot(arm(n-1) $ rot2(spiral(n-1))))
        when n > 1;
```

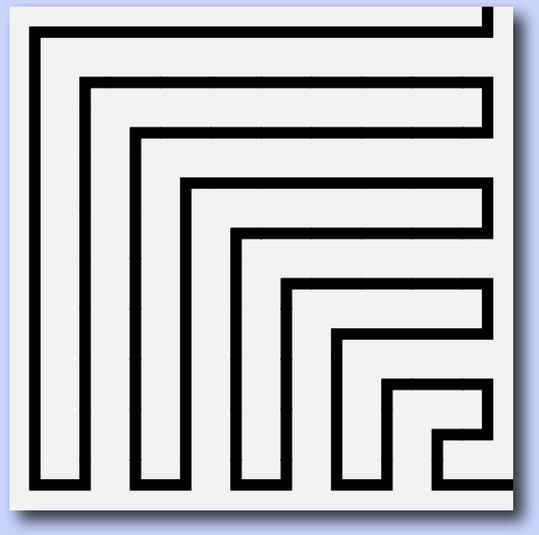


The results

spiral(10)



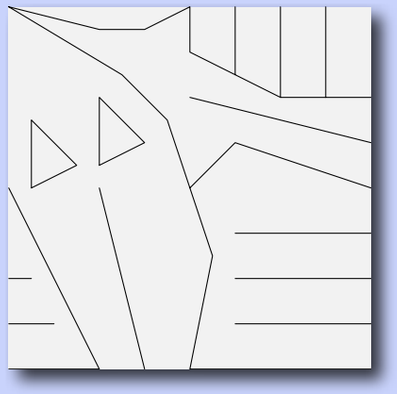
zagzig(10)



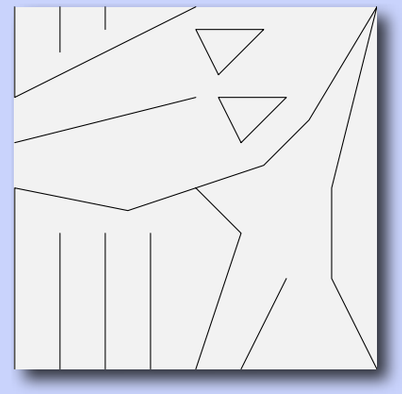
The big challenge

Take four square tiles:

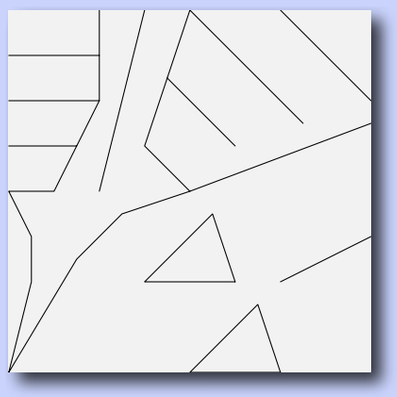
A



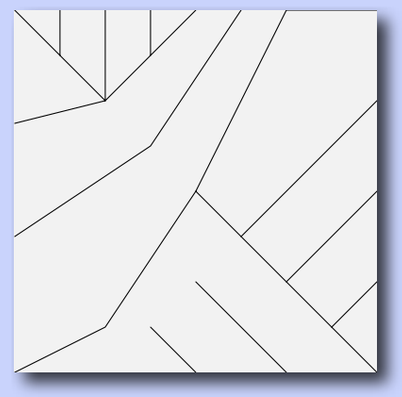
B



C



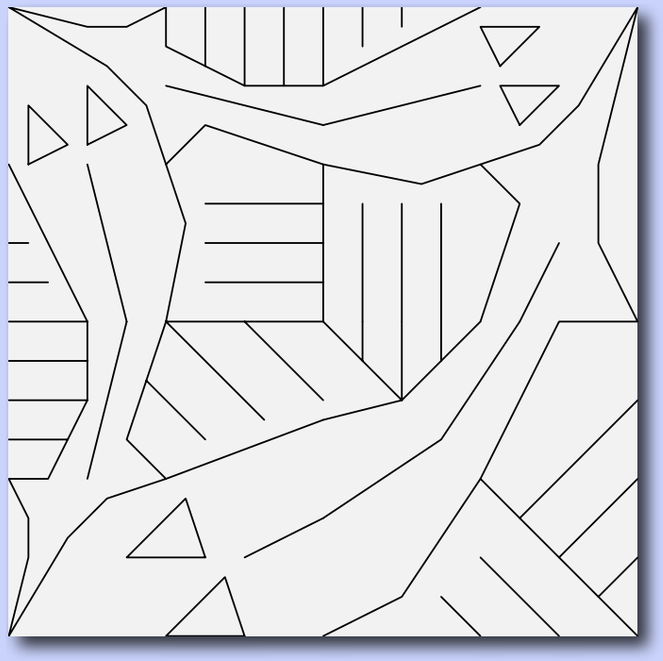
D



The big challenge

They fit together to make a bigger square:

define T =
(A $ B) & (C $ D);



The big challenge

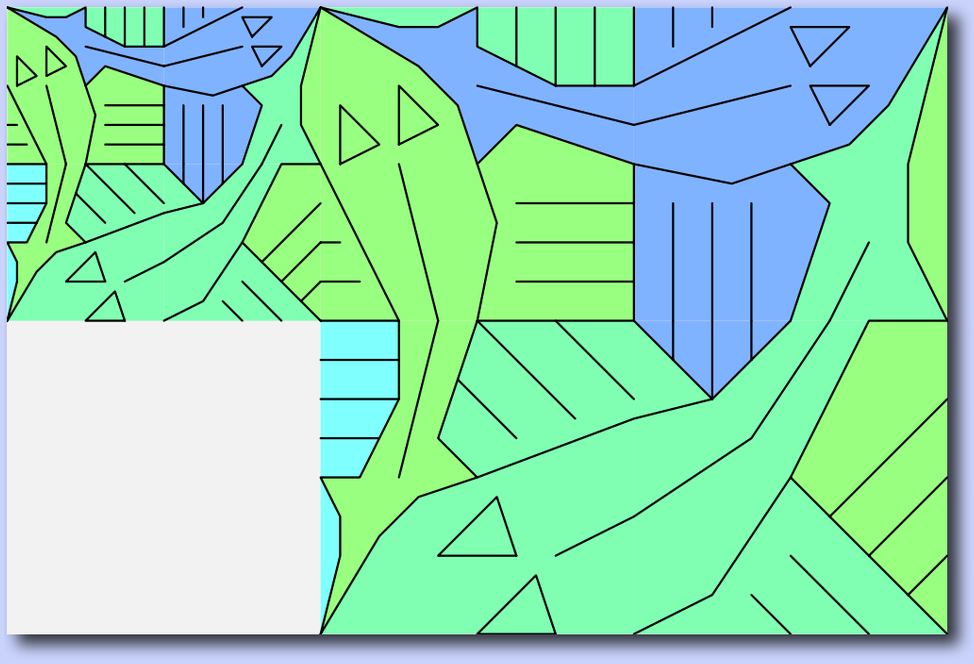
… and a copy of that tile fits next to itself like this:

(T & blank) $ T

The big challenge

… and a copy of that tile fits next to itself like this:

(T & blank) $ T



The big challenge

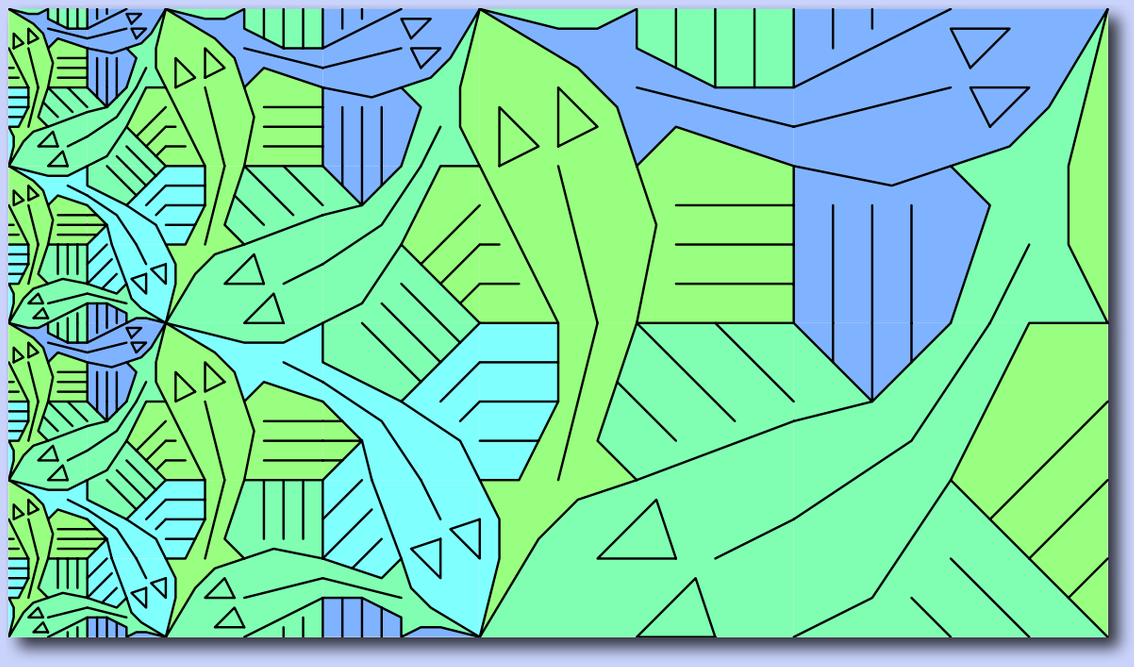
… and another copy fits here:

(T & rot(T)) $ T

The big challenge

Then we can add another layer:

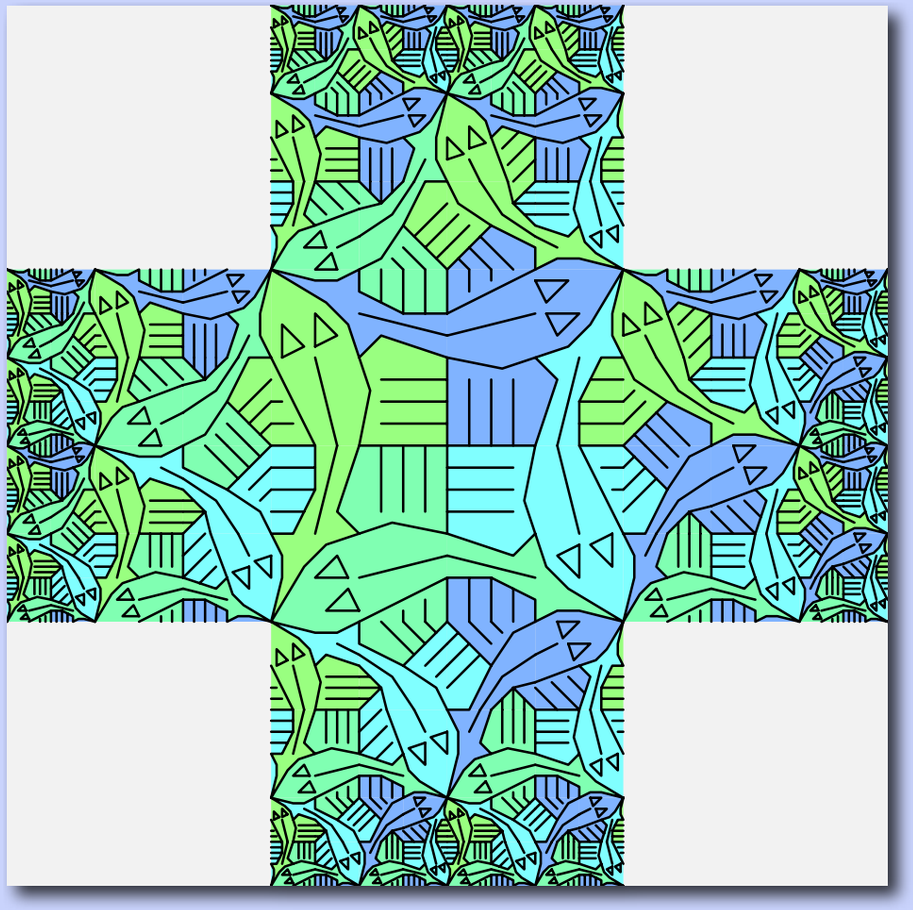
```
side(2) $ T
```



The big challenge

… and arrange four copies around a centre:

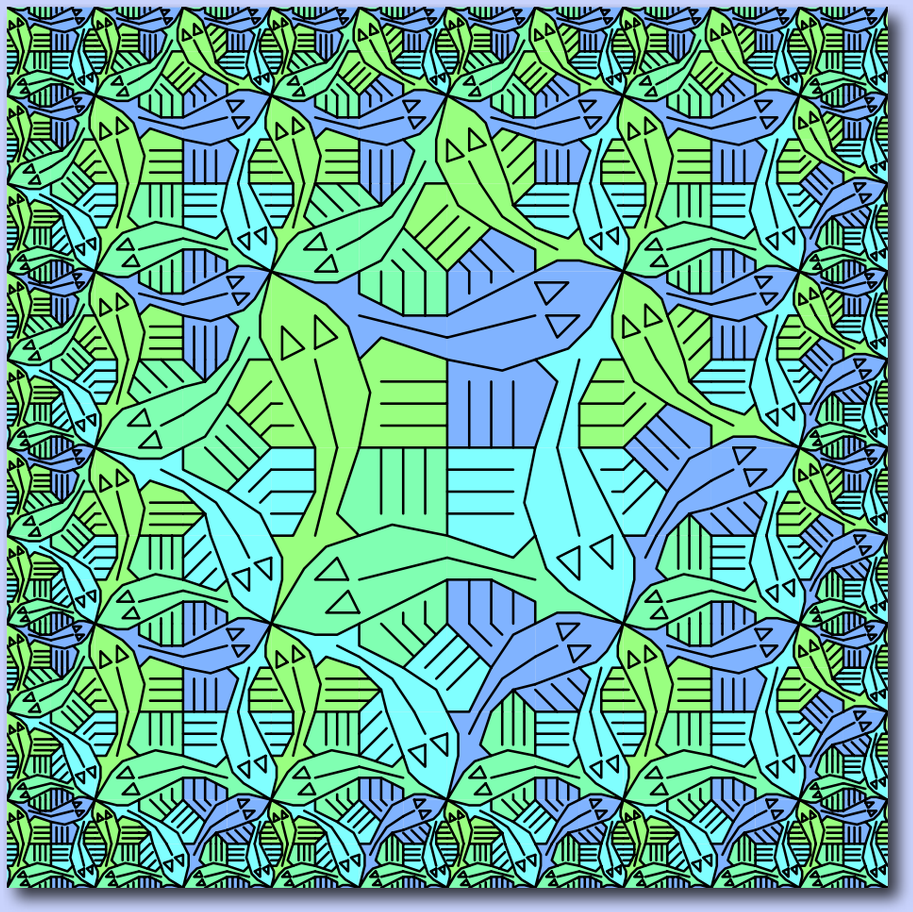
frame(blank, side(2), U)



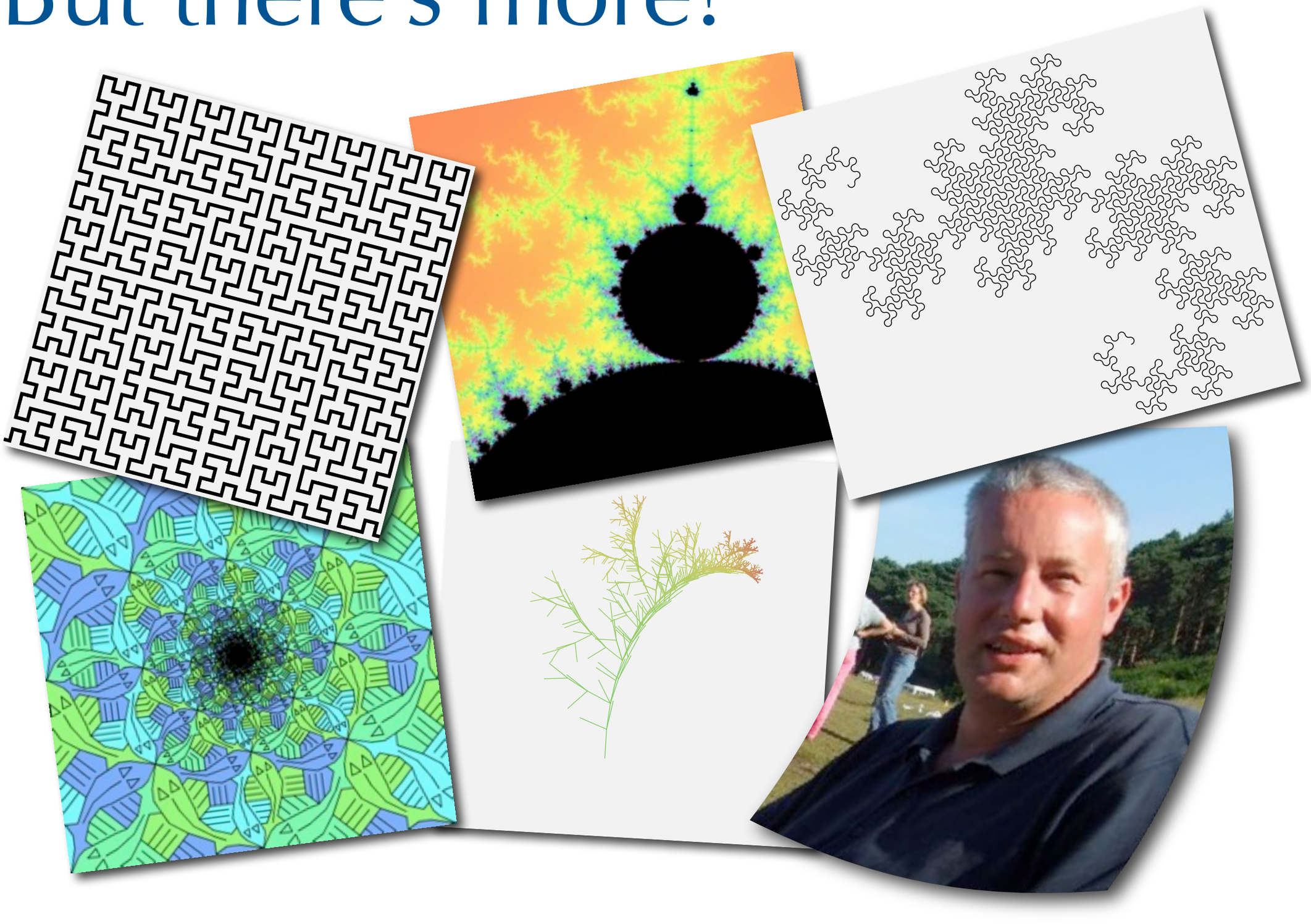
The big challenge

Finally, find something to fill in the corners:

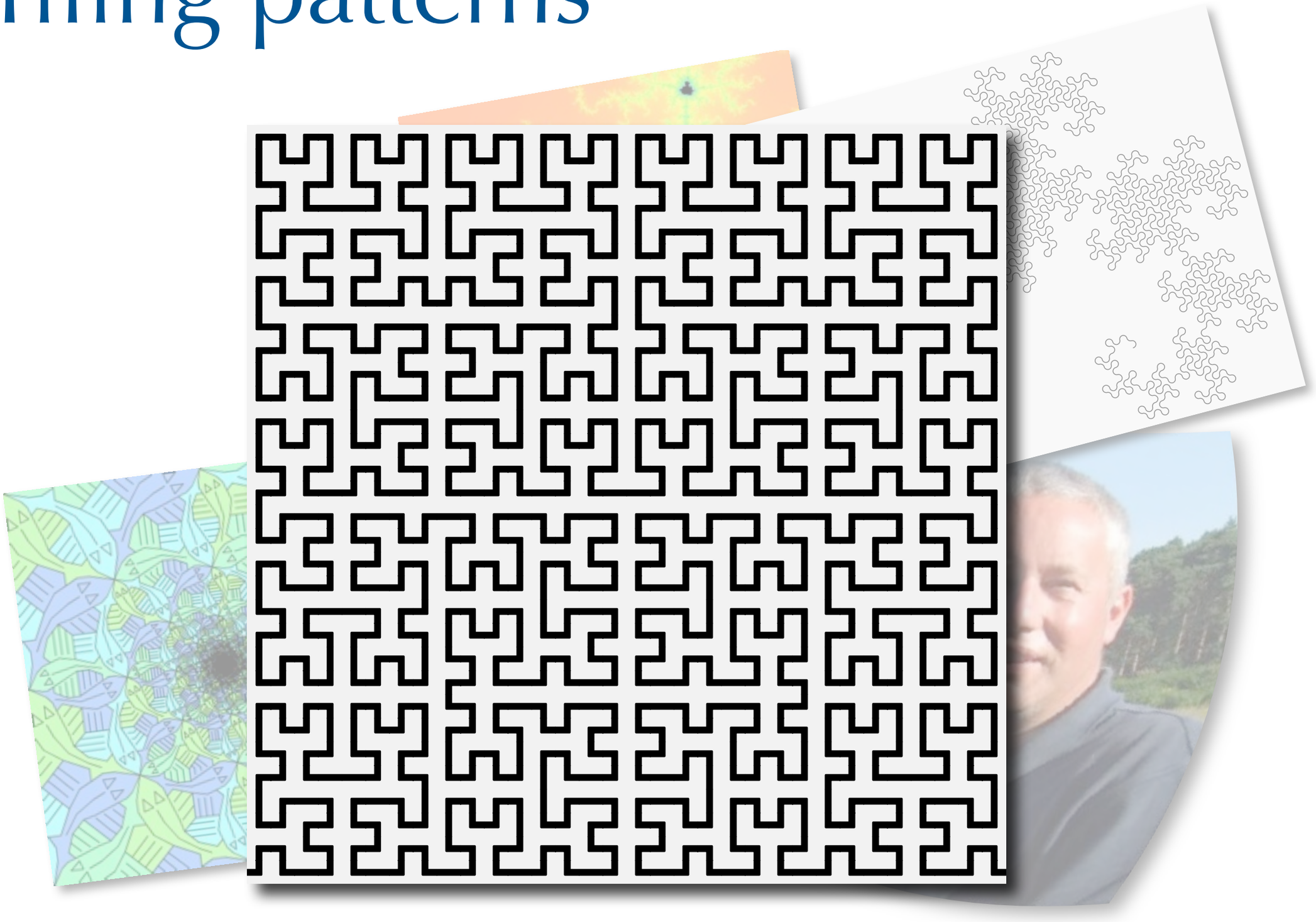
frame(corner(2), side(2), U)



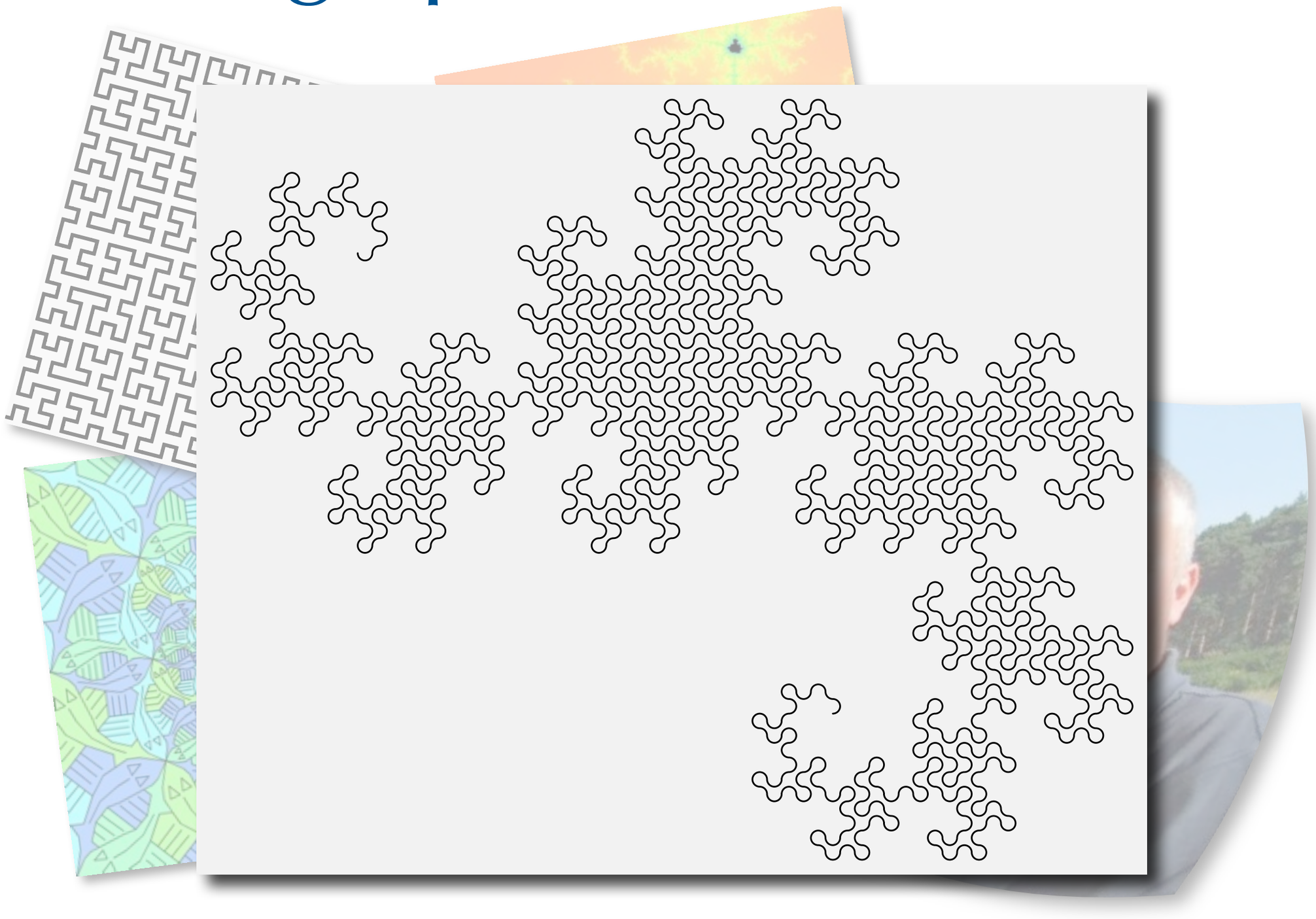
But there's more!



Tiling patterns



Turtle graphics



Fractal plants

Pixel images

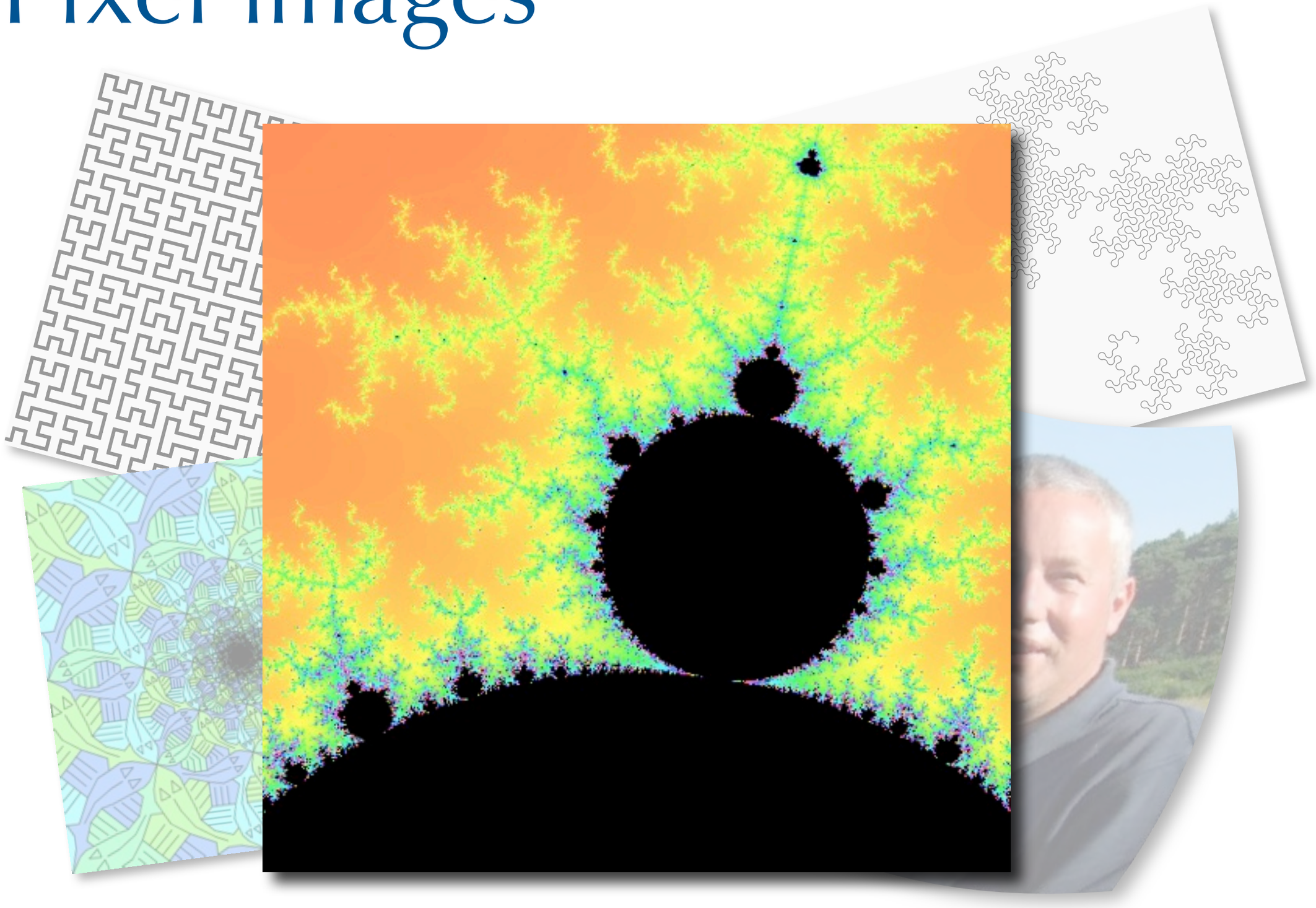
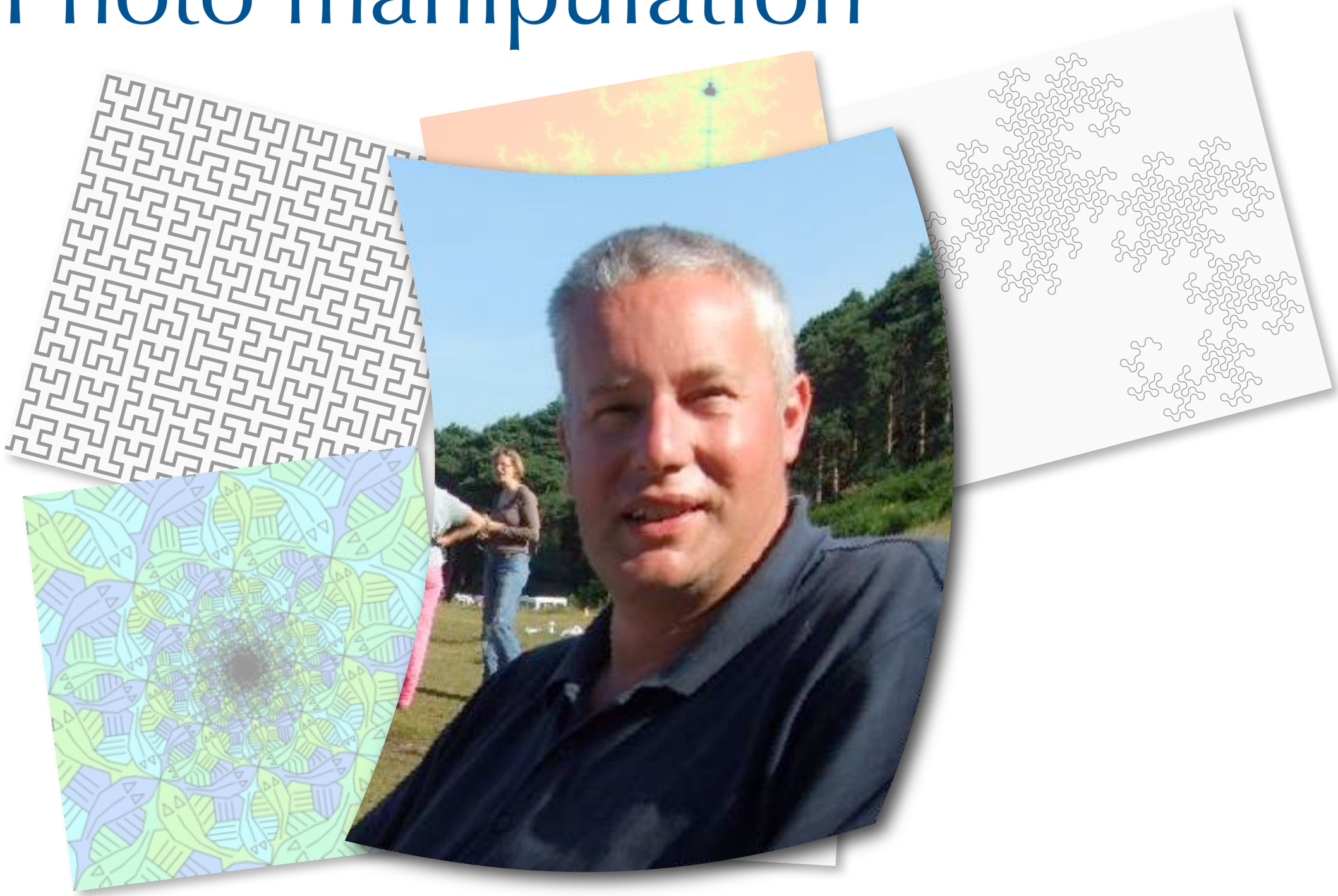


Photo manipulation



Images as functions

We can represent a photo as a function that maps coordinate pairs to colours:

mike([150, 200]) = rgb(0.96, 0.68, 0.65)

Then image(300, 400, mike) draws the picture by evaluating the function at 120 000 points.

Transforming the output

Define a function *weird*:

```
define weird(rgb(r, g, b)) = rgb(r, b, g);
```

Then *weird* . *mike* represents a picture where green and blue are swapped.

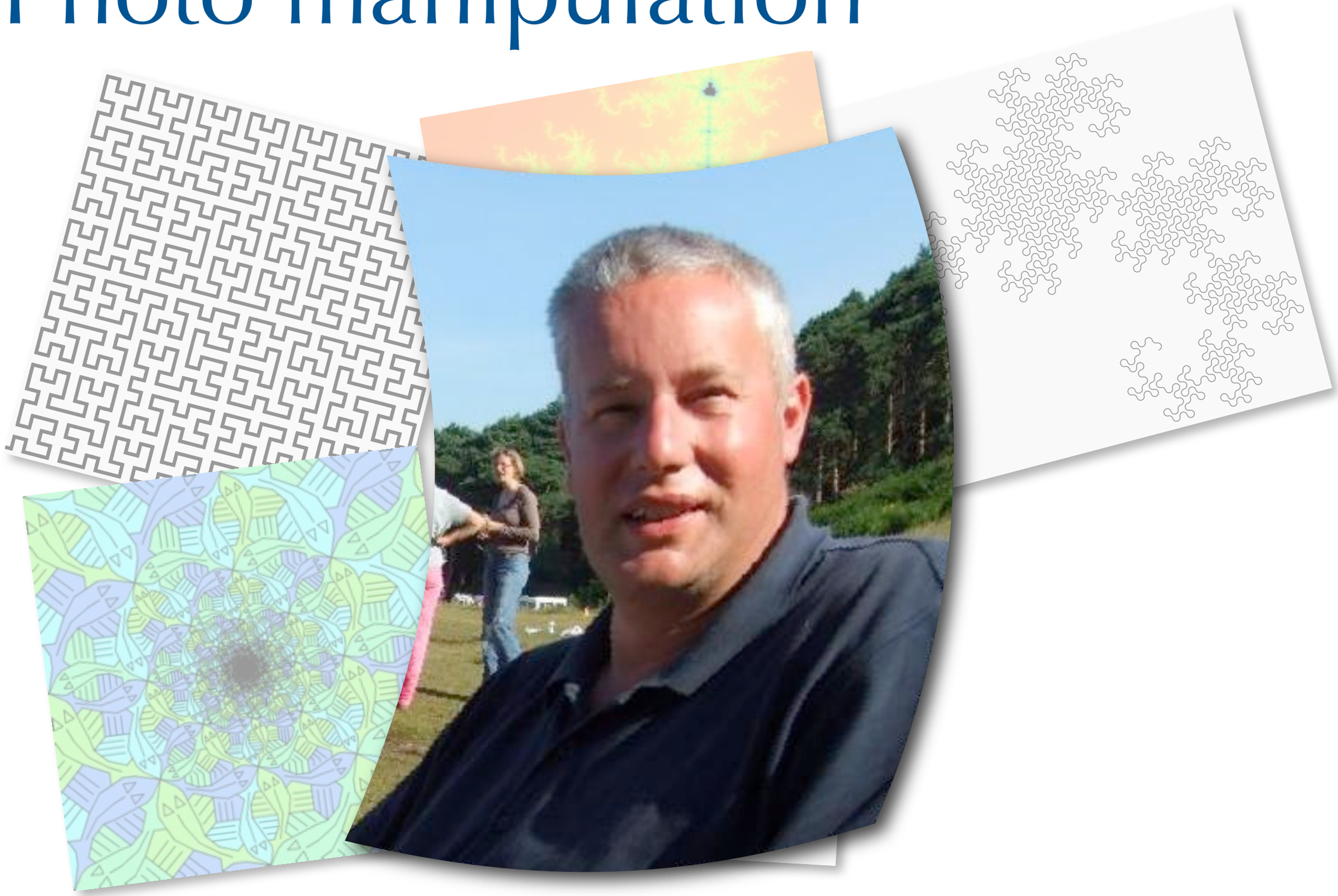
Transforming the input

Define a function *shear*:

```
define shear([x, y]) = [x-y/5, y];
```

Then *mike . shear* represents a picture that has suffered a linear transformation.

Photo manipulation



Animations

Functional programming

- No ‘programming’ variables – use mathematical variables instead.
- No loop commands – use recursion instead.

Two benefits:

- Easier to make programs from independent pieces.
- Easier to reason about them mathematically.

Abstract data types

A set of values with operations on them.

- The operations obey rules that can be described algebraically.
- The computer representation of values is hidden.

Abstract data types

Example: pictures with $, &, rot.

- rot(p $ q) = rot(q) & rot(p)
- aspect(rot(p)) = 1 / aspect(p)

Pictures are *represented* using coordinates and transformation matrices in a way that is hidden.

Enriching school maths

- There's more to algebra than "x is the unknown".
- There's more to recurrences than "guess the next number".
- May we dream of a world where survival depends on mathematics?

The GeomLab site

`http://www.cs.ox.ac.uk/geomlab`

The software:

- runs from the web page.
- Java-based – no installation required.

Teaching materials:

- full set of worksheets for a 1-2 day activity.